CENG313 **Fall 2018**

PROGRAMMING ASSIGNMENT 4

Due date: 14.12.2018 10:00 am

Write a Pthreads program that implements a “task queue.”

The main thread begins by starting a user-specified number of threads that immediately go to sleep in a condition wait (the worker threads are idle at first).

The main thread generates tasks to be carried out by the *other* threads. A task is either an insert, a delete, or a search a value in a sorted list (in ascending order) with no duplicates. Each time the main thread generates a new task by adding it to a task queue, it awakens a thread with a condition signal. When a thread is awakened, it pulls a task from the queue and processes it. When a thread finishes executing its task, it should return to a condition wait. When the main thread completes generating tasks, it sets a global variable indicating that there will be no more tasks, and awakens all the threads with a condition broadcast.

To implement the above system, you should use Pthreads threads, mutexes, and conditions (not the semaphores) as covered in the class.

You are free how to implement your task queue. But (if you prefer) you can use the template given as taskqueue.c file with the following data structures and C functions (similar to the Programming Assignment 2):

```
/* Struct for list nodes */
struct list_node_s {
   int    data;
   struct list_node_s* next;
};

/* Struct for task nodes */
struct task_node_s {
   int task_num;
   int task_type; // insert:0, delete:1, search:2
   int value;
   struct task_node_s* next;
};

/* List operations */
int Insert(int value);
int Delete(int value);
int Search(int value);


/* Task queue functions */
void Task_queue(int n); //generate random tasks for the task queue
void Task_enqueue(int task_num, int task_type, int value); //insert a new task into task queue
int Task_dequeue(long my_rank, int* task_num_p, int* task_type_p, int* value_p); //take a task from task queue
```

Your program will get the number of threads and the number of tasks from the user, and display the task generation and completion information (in arbitrary order). In the final phase, it should print the contents of the list.

Example execution:

```
./queue 2 10
Thread 1: task 0: 3 is inserted
Thread 0: task 1: 17 is inserted
Thread 0: task 3:  6 cannot be deleted
Thread 0: task 4: 9 is inserted
Thread 1: task 2: 13 is inserted
Thread 1: task 6:  9 cannot be inserted
Thread 1: task 7: 3 cannot be inserted
Thread 1: task 8: 0 is inserted
Thread 1: task 9: 12 is inserted
Thread 0: task 5:  2 cannot be deleted
main:  Final list:
  0 3 9 12 13 17
```

As part of the assignment, you need to run a set of experiments and evaluate the performance of your parallel implementation. For this part, you need to have the sequential version (the version in the Programming Assignment 2) to compute the *speedup*, as $S= T_S/T_P$ where T_S is the execution time of the sequential version, T_P is the execution time of the multithreaded implementation.

You will first execute the sequential version (not the execution with 1 thread!) for different number of tasks (1000, 10000, 100000), then run your multithreaded program with various number of threads (1, 2, 4, 8, 16) and tasks (1000, 10000, 100000), and compute the speedup for each execution. You are required to plot a graph (speedup/number of threads) for various task numbers indicating the performance of your implementation. You will have three graphs which are for different task number-executions (1000 tasks, 10000 tasks, 100000 tasks).

Notes:

- You need to use an appropriate timing function (as discussed in the lab) to determine the execution time of your program.
- You need to work individually, no group work is allowed.
- No late homework will be accepted.
- You are required to submit a report that includes implementation details, screenshots of your sample executions (with small number of tasks), graphs, your observations about the performance of your implementation, how you interpret the results.

Submission: You are required to submit your **commented** source code and report to CMS. Please create a compressed file including all source files and report; and name it as yourstudentnumber_P4.zip (e.g. If your student number is 201812345678_P4, the file name must be 201812345678_P4.zip).